Geophysica (1994), 30, 1-2, 93-105

Lossless Compression of Interpolated and Raw MRL-5 Weather Radar Data

J. Näppi

University of Turku, Department of Computer Science
Lemminkäisenkatu 14 A, FIN-20520, Turku Finland

(Received: July 1994; Accepted: November 1994)

Abstract

Experimental results of lossless compression of MRL-5 weather radar data used at the Finnish Meteorological Institute are presented. The data are usually sparse, and can thus be compressed efficiently. Simple run-length algorithms reduce the required storage usually by more than 80 %. When the run-length code is compressed with Huffman-coding or arithmetic coding, additional 20 % compression is gained. The digitization noise and sparseness of the data make further significant compression hard without resorting to lossy techniques.

1. Introduction

The Finnish Meteorological Institute (FMI) has been operating digital weather radar network in Finland since 1985 (*King*, 1989). The radar observation data is compressed losslessly for archiving purposes. Lossless compression means that the original data are transformed to a representation requiring less storage space, and the compression can be reversed (expansion) to produce exactly the original data, without any loss, from the compressed representation.

To evaluate the compression achieved on a certain (data) file, a suitable measure is needed. In this article, *compression factor* is used. It is defined as

$$\theta = \frac{l(s)}{l(t)}$$

where $l(s)$ is the space, expressed in bytes, required to store the original file s, while $l(t)$ is the space required to store the compressed file t. For example, if $l(s)$=57600 (bytes) and $l(t)$=5760 (bytes), the compression factor is $\theta = 10$.

To reduce the amount of backup tape needed, only parts of the original raw radar observation data have been archived at FMI. In this article, compression of three types of such data files is considered: raw observation data scanned from the lowest radar elevation

angle, height data levels interpolated from the original raw data, and maximum echoes in vertical direction (FMI has started archiving the complete 3-dimensional reflectivity data since Summer 1994, due to improvements in the weather radar equipment).

Few compression tests involving weather radar data have been published. For example, (*Puhakka et al.*, 1979) describes three (combinable) algorithms, and results on compressing weather radar data using those algorithms. Sometimes weather radar data has been used to test compression algorithms (*Leung et al.*, 1991). Recently there has been interest to compare WMO BUFR compression with alternative techniques (*Dietrich et al.*, 1990) (*Newsome*, 1992). BUFR is a general format to represent meteorological data. It has been noted that run-length algorithms and Lempel-Ziv algorithms (section 3) seem to compress better than BUFR. Furthermore, experiments in routine weather radar operation at FMI show that high compression factors are achievable with a relatively simple run-length algorithm (section 3).

Although weather radar data formats vary, the data are commonly quite sparse, and hence the compression factor is usually high, ranging at FMI on average from 7.0 to 37.0 (*King* , 1989)(*King* , 1993). Since compression of satellite and SAR images for example is usually considerably harder, more effort has been put on that subject (for example (*Chen et al.*, 1987) (*Hadenfeldt et al.*,1994) (*Chang et al.*, 1988) (*Memon et al.*, 1994)).

In this article, it is examined how much the compression of the run-length method used at FMI can be improved, using algorithms with about the same running time (to maintain the transparency of compression). These algorithms can also be applied to other kinds of sparse data. In section 2, test data used are described. The algorithms are presented in section 3, and the compression results are presented in section 4.

2. *Test data*

The MRL-5 weather radar data files used in the tests were supplied by FMI, and were selected as follows (the MRL-5 radars are of Soviet construction (*King*, 1989)). A number of randomly selected data files were visually examined as two-dimensional images on terminal. From these, 192 data files were selected so that they included sparse cases (few or no clouds) as well as more active cases (large clouds or cloud formations filling the radar observation radius), and cases in between (like in Fig. 1).

There were three types of data files: IPPIN, IC and IHMXN. All contain planar data in 240*240 pixels, a pixel representing radar reflectivity as eight bits (i.e. a byte). To simplify the discussion, the term *pixel* is used to denote a single unit of weather radar data although such units could also be called *echoes*, *radar reflectivity*, or *symbols*, depending on the context.

IC-files are collections of 12 height data levels, each level containing (interpolated) observation data at a certain height. The lowest height is 500 meters, and the vertical distance between adjacent levels is 1 kilometer. The levels are generated from the original raw radar observation data using a 4-point interpolation scheme (*Näppi*, 1994, p. 29) (*King*,

1993). The horizontal and vertical distance of adjacent pixels on any level is 2.5 kilometers (the nominal maximum range of the radars used is 300 kilometers (*King*, 1989)).

Fig. 1. A weather radar data image.

IHMXN-files relate to a certain IC-file. Each pixel position (x, y) of an IHMXN-file contains the greatest pixel on any IC-data level in the same position (x, y). So an IHMXN-file contains the maximum echoes of the levels of an IC-file. IPPIN-files contain original raw observation data, scanned from the lowest radar elevation angle (0.30 degrees).

The radar software (Ericsson Weather Information System) produces the data files automatically from the original raw radar observation data represented in spherical coordinates (*King*, 1989). The pixels which represent radar reflectivity under a specified threshold value (-30 dBZ) are assigned the default background value 0, as are pixels outside the range of the radar. The background value will be called *background pixel*. More details of the measurement system and the test data can be found in (*Näppi*, 1994) and (*King*, 1989).

The data files were further reduced from the 192 files to 82 files by considering their *order 0 entropy* (entropy for short) (*Shannon*, 1948) (*Bell et al.*, 1990)

$$H_o(f) = -\sum_{i=0}^{255} p_i \log_2 p_i$$

where p_i is the occurrence frequency of pixel i in file f. The unit of entropy is bits per pixel. If several files of the same type (e.g. IHMXN) had about the same entropy (like 0.66 and

0.67 bits per pixel), only one of these files was used. Files with the same entropy tend to compress similarly, although one can find (theoretical) exceptions.

Entropy of the test files was quite low (Table 1). This is because of the large proportion of the background pixels in the data files.

The digitization noise of the least significant data bits introduces randomness hard to predict (Fig. 2). Since the data are also sparse, useful pixel occurrence statistics are hard to gather. A particular pixel ε is difficult to predict accurately using the neighbour pixels, unless they are background pixels, in which case ε is probably a background pixel.

Table 1. Information on the test files. The average entropy over all test files was 0.76 bits per pixel. p(β) is the ratio of the amount of the background pixel to the amount of other pixels.

Info	IHMXN	IPPIN	IC
amount	10	36	3 * 12
min. entropy	0.33	0.63	0.00
av. entropy	1.21	1.02	0.37
max. entropy	1.71	1.27	1.40
$p(\beta)$	0.84-0.98	0.89-0.95	0.87-1.00
max. pixel	94-160	148-184	0-172

78 43 41 25 65 70
73 56 53 64 65 73
72 44 53 71 72 70
78 78 76 76 70 74
77 38 85 75 73 72
69 57 84 73 72 76

Fig. 2. A 2-dimensional sample of the data (IHMXN). Small variations in the data make accurate prediction of a certain pixel difficult.

3. *Algorithms*

The current method (*Sandstroem*, 1986) at FMI to compress weather radar data is based on run-length coding (*Reghbati*, 1981) (*Lu*, 1993). The data file is scanned row-wise and compressed into groups (H,D) with header and data part. The header part H has either one ($H=b_1$) or two ($H=b_1b_2$) bytes (Fig. 3). Two bits in b_1 indicate the length of the header (i.e. one or two bytes) as well as the contents of the data part (i.e. run-length group or uncompressed pixels). The other bits of the header byte(s) indicate the length of the uncompressed string the group represents. If the data part D is compressed it contains only one byte: the repeated pixel. Otherwise the data part consists of pixels repeating at most three times successively. A pixel repeating at least four times is more efficiently compressed using a run-length group. One data group can represent at most 63 ($H=b_1$, 6 length bits) or 16383 ($H=b_1b_2$, 14 length bits) bytes. This algorithm is called *rlew*.

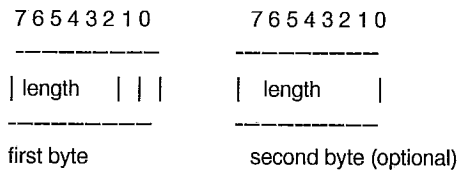

Fig. 3. The group header format used in EWIS (*Sandstroem*, 1986).

A slightly better compression factor is obtained by using two special *esc*-pixels, which may be for example pixels not occurring in the data, or the least frequently occurring pixels. This is a common compression trick. Normally a pixel ε is encoded as itself. However, *n* successive repetitions of ε are encoded as triplet (*esc*,*n*,ε). For representing *n*, either one ($esc=esc_8$) or two bytes ($esc=esc_{16}$) are used. This algorithm is called *rl816* . Two unused pixels were defined as *esc*-pixels in *rl816* .

As an example, consider a number series like 4,4,4,4,4,5,3,6,300x4. Using *rl816*, the series is encoded as (esc_8,5,4),5,3,6,(esc_{16},300,4).

Plain Huffman-coding (*Abramson*, 1963) (*Lelewer et al.*, 1987, 271-274, 278-283) is not itself suitable for weather radar data compression since every pixel requires at least one codebit - an original file of 57,600 bytes is compressed to 7,200 bytes at best. Better results are achieved by using run-length coding first, and compressing the resulting code with Huffman-coding. Hence two kinds of redundancy are handled: run-length coding reduces successive repetitions of pixels, while Huffman-coding makes use of the non-uniformity of the pixel frequencies. This algorithm is called *rlh0*. For the run-length part, *rl816* is used since it produces shorter code than *rlew*, and since the *rlew*-code isn't more compressible (this was tested separately (*Näppi*, 1994, p. 41)). Furthermore, *rl816* is simple to implement.

The Huffman-coding part uses the traditional *semiadaptive order 0 model* (*Lelewer et al.*, 1987, 271-274). That is, the code is scanned twice: first the pixel weights (the occurrence counts of the pixels) are computed, and then the data are encoded using the weights. The code is further compressed by saving the weights in groups (*i,j,W*), where i indicates the first and *j* the last pixel index of the weights in W. So if pixels 4 - 16 and 140 - 254 do not occur in the file, the weights can be encoded into groups (0,3,w_0..w_3), (17,139,w_{17}..w_{139}) and (255,255,w_{255}). A group may include zero-valued weights, if they occur at most three times successively.

rla0b-algorithm is similar to *rlh0* except that instead of Huffman-coding, arithmetic coding (*Rissanen*, 1979) (*Witten et al.*, 1987) is used. An additional enhancement with respect to *rlh0* is to restrict the run-length coding to the background pixel only. The compression improves since the repeating pixel can be omitted in run-length triplets, and since there are practically no other repeating pixels than the background pixel. This enhancement could also be added to *rlh0*, but this way we get more information about

enhancement could also be added to *rlh0*, but this way we get more information about combining run-length coding with aftercoding. For example, if *rlh0* would likewise omit the repeating pixel, the resulting compression factor would be between the results of the original *rlh0* and *rla0b*.

To make use of the evident two-dimensional dependencies between the pixels, hierarchical image partition methods (like *quadtree* (*Samet*, 1984)) were examined (*Näppi*, 1994, 46-48). Unfortunately regular-formed partition blocks have quite a few background pixels which tend to reduce the compression factor. Although blocks could further be encoded using run-length coding, it is then more efficient (with respect to compression factor and running time) to use run-length coding only, without additional image partitioning. Therefore an algorithm similar to method of Howard and Vitter (*Howard* and *Vitter*, 1991) was tried.

In this algorithm, repetitions of the background pixel β are run-length encoded as in *rl816*. Other symbols $X \neq \beta$ are encoded by making a prediction $\tilde{X}$ for X. The prediction error $\Delta = X - \tilde{X}$ is encoded as described below.

The value of X is predicted using four neighbour pixels of the current pixel context $\Psi = \Psi(A,B,C,D)$ (Fig. 4). Various prediction methods were considered from the ones in lossless *jpeg* to simple perceptron algorithms (*Näppi* 1994, Appendix B). Best results were achieved by using a small collection of simple (linear) predictors simultaneously. Each predictor i has cumulative score counters s_{ij}, where $j = 1$ to 16 refers to a certain prediction context, based on presence of the background pixel β in Ψ. For example, $j = 1$ is a pixel context where there is no β, $j = 2$ is a context where β occurs in position B, and $j = 16$ is a context where β occurs in every pixel position of Ψ. Before the coding, all counters s_{ij}

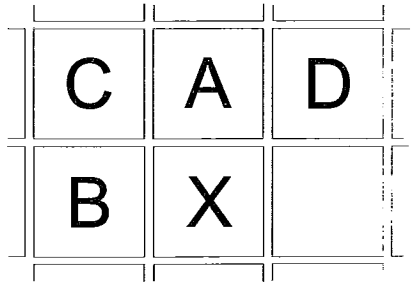

Fig. 4. The pixel prediction context Ψ used in *prl816*. X is the pixel to be encoded.

are initialized to 0.

When coding a pixel X, the current prediction context j is determined. Each predictor i then makes an estimate and produces a prediction $\tilde{X}_i$. If the prediction is correct, Υ_1=1 points are added to s_{ij}. If the prediction is better than anyone of the other predictors, $\Upsilon_2 = 1$ points are likewise added to s_{ij}. Other (integer) parameter configurations (Υ_1, Υ_2) were also

tried considering various sets of data files, predictors and prediction statistics of the predictors, but the results were not so good as with the parameters above. It was usually sufficient to use only two simple predictors (i.e. these consistently got the highest scores), the other predicting the background pixel β and the other predicting the previous pixel B. The latter is used primarily inside cloud formations, while the former is used near cloud borders and pixel contexts containing many background pixels.

The prediction error $\Delta = X - \tilde{X}$, produced by the predictor with highest s_{ij}, is encoded by using Laplace distribution

$$f_{\mu,\delta^2} = \frac{1}{\sqrt{2\delta^2}} \exp\left(-\sqrt{\frac{2}{\delta^2}} \, | x-\mu | \right) \tag{1}$$

Here μ is 0 (perfect prediction). The variance of the distribution (δ^2) is determined by the error statistics gathered from the previous occurrences of Ψ. For example, if the predictions in a certain context are close to 0, the variance should be small. Poor predictions increase the variance, raising the coding probability of pixels far away from 0. However, if there haven't been enough occurrences for a reliable estimate, search for the error statistics is repeated by ignoring the least significant bit(s) of pixels in Ψ, and considering the statistics of the pixel contexts found in this way. So *approximate context modelling* is used. For further details, see (*Howard* and *Vitter*, 1991).

Run-length algorithms compress long runs of the background pixel very efficiently. On the other hand, there are hardly any other repeating pixels or strings of pixels in the data. Hence, algorithms based on Ziv-Lempel -methods (*Ziv et al.*, 1977) (*Ziv et al.*, 1978) rarely performed better than run-length algorithms (*Näppi*, 1994, p. 44). *gzip*, *compress*, *pkzip* and two experimental algorithms (based on *lzw*- and *lz77*-algorithms (*Nelson*, 1991)) were tested. Only the results of *gzip* are presented since it performed generally better than the others. The algorithm of *gzip* is based on fusion of *lz77* and Huffman-coding (as is *pkzip*, *compress* is a *lzw*-algorithm based on *lz78*).

Other algorithms were also considered (*Näppi*, 1994, 39-48), but their performance was (generally) significantly worse than that of the algorithms discussed above. They included lossless *jpeg* (version 1.1, freely distributable by Portable Video Research Group at Stanford) (*Wallace*, 1991) which had bad compression factor, change of the image scanning order (generally no advantages), as well as variations of the algorithms discussed above. Some experiments were also made by using an *adaptive order n model* (*Cleary et al.*, 1984) with arithmetic coding, originally developed for text compression. Best results were usually gained by limiting the highest order to 1 (because of the rapid decrement of interpixel dependencies as a function of distance). The compression results were about the same as with the other test algorithms, but the time complexity was unacceptably high (about 10 times the others).

The algorithms considered here are listed in Table 2.

Table 2. The compression methods.

Program	Basic method
rlew	run-length
rl816	run-length (two *esc*-pixels)
rlh0	run-length + Huffman-coding
rla0b	run-length + arithmetic coding
prl816	run-length + approximate 2D coding
gzip	*lz77* + Huffman-coding

4. *Results*

The algorithms (except *gzip*) were tested in Vax/Vms-environment using a Vax-4600 computer (32 MIPS), and implemented in Vax-C 3.2. The algorithms were not particularly optimized for speed (e.g. no analysis of the machine language code) although inefficient code was removed. *gzip* (by Free Software Foundation) is a separate, freely distributable program, tuned for good general performance.

Average compression results for each data file type (described in section 2) are listed in Table 3. The run-length algorithms *rlew* and *rl816* form one group while the other algorithms compress about 20 % more. The absolute differences are small, however, so the advantage of improved compression is noticeable only when compressing vast amounts of data.

Table 3. Compression results (as compression factor). A file containing only background pixels is compressed from 4 to 14 bytes, except with *gzip* which used 106 bytes.

Program	IHMXN	IPPIN	IC
rlew	7.25	8.08	21.14
rl816	7.42	8.27	21.83
rlh0	8.65	9.73	25.84
rla0b	8.90	10.01	26.95
prl816	9.13	9.07	26.18
gzip	8.61	9.51	25.13

Table 4 contains average results for *rla0b* when compressing IC-data levels. The compression factor follows the entropy of the levels.

The cpu-time results (Table 5) are averages of 25 compression and expansion (decompression) runs. The results for *gzip* are omitted since, at the time of the tests, a working version of *gzip* was not available in Vax/Vms-environment (*gzip* was tested in a Unix system). However, *gzip* seemed fast, and anyway comparison with the other algorithms would not be fair because of the more efficient implementation.

Table 4. Average compression factor (acf) of various IC-levels when using rla0b, compared to their average entropy.

IC-level	acf	Entropy
level 1 (lowest)	86.40	0.10
level 2	29.84	0.37
level 3	11.21	0.95
level 4	8.39	1.21
level 5	9.64	1.03
level 6	17.96	0.54
level 7	45.63	0.21
level 8	142.90	0.06
level 9	478.67	0.01
level 10	1136.84	0.00
level 11	1710.89	0.00
level 12	2742.86	0.00

Since *rlew* is the compression algorithm currently in use at FMI, it is used as a reference method (Fig. 5). All the other algorithms compress better, although note that *rlew* is somewhat more error-tolerant (errors in data groups need not accumulate since the length of a run-length group is known). The complicated compression requires more cpu-time than with the other algorithms (*prl816* as an exception), but the expansion is fast due to its simplicity. More efficient implementation might increase the compression speed close to that of *rl816*. *rl816* compresses slightly better than *rlew* because of the more efficient coding of the background pixel. Compression and expansion of *rl816* are very similar and take about the same time.

Table 5. Average cpu-time usage (compression/expansion) in kilobytes/second.

Program	IHMXN	IPPIN	IC
rlew	1450/2950	1500/2800	1550/2950
rl816	3300/2800	3300/2800	3500/2950
rlh0	1950/1900	1950/1900	2700/2450
rla0b	1600/1100	1750/1200	2450/1950
prl816	500/450	600/500	1100/1050

The other test algorithms have better compression factor because of the additional encoding of the run-length code. However, being more complex they take more time.

The Huffman-coding used in *rlh0* is reputedly fast. Expansion of Huffman-code is simpler than compression, but the compression and expansion take still about the same time. This is because the run-length code to be compressed is so short.

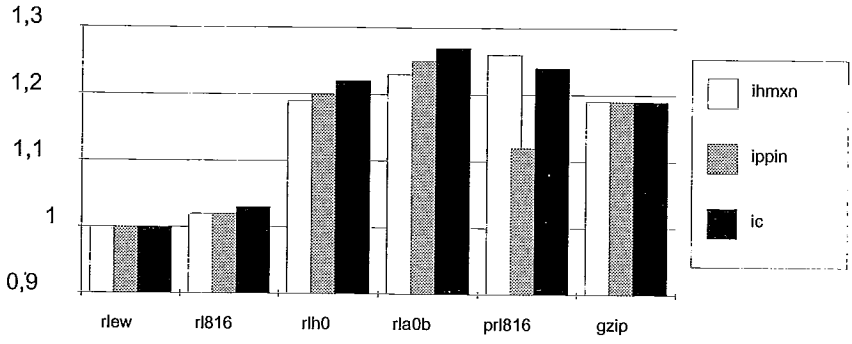

Fig. 5. Relative compression factor with respect to *rlew*. Relative factors for IHMXN, IPPIN and IC are presented.

rla0b compresses better than *rlh0* because of the arithmetic coding. However, arithmetic coding demands more cpu-time when compared to Huffman-coding. Also, if other pixels than the background pixel are encoded in the run-length part (like in *rlh0*), the absolute difference to *rlh0* is only few dozens of bytes at most. So compression of the run-length code with Huffman-coding or arithmetic coding leads to similar results. Faster multiplication-free implementations of arithmetic coding (*Rissanen et al*, 1989) (*Chevion et al*, 1991) would evidently reduce the compression factor difference between *rlh0* and *rla0b* even further.

prl816 is a highly experimental algorithm. The added complexity requires more cpu-time than with the other algorithms. However, some potential is evident. For example, sparse data files (less than 200 observations) compress well: since the (often singular) pixels are close to the background pixel, Laplace distribution gives them good probabilities. *prl816* was the only algorithm to make use of the two-dimensional dependencies between pixels. Unfortunately the data are often so sparse that it is hard to collect useful coding statistics. More sophisticated error modelling (*Howard* and *Vitter*, 1992) and adjustment of the coding parameters might improve the compression.

Since there are hardly any repeating strings (other than strings of the background pixel) in weather radar data, *gzip* works almost like *rlh0*. However, because *gzip* has to "learn" the run-length algorithm, and because of the more complex coding format of *gzip*, *rlh0* usually has a bigger compression factor. The simplicity of run-length coding makes *rlh0* also faster (at least theoretically), and simpler to implement. Adjustments like preloading the dictionary with common data strings might increase the usefulness of Ziv-Lempel methods, although it is questionable if the implementation is worth the effort. An explicit form of run-length coding should probably be included in Ziv-Lempel implementations.

IHMXN-files are harder to compress than the other files since they contain the least number of background pixels. IC-files contain lots of background pixels and are thus most compressible. IPPIN-files contain a great proportion of the echoes of an observation volume, and can be compressed only marginally better than IHMXN-files. Thus the amount of the background pixel seems to influence the compression most. However, *prl816* compressed IHMXN-files slightly better than IPPIN-files. The reasons for this might be that IHMXN-files contain similar echoes with respect to those in IPPIN-files, there are more echoes to be used in coding statistics, and the difference between adjacent echo values (pixels) is not as high as with IPPIN-files. However, definite conclusions cannot be given due to small data sample set.

5. *Discussion*

The results were somewhat expected. Long runs of the background pixel can be most efficiently compressed by using run-length coding while the other pixels are best compressed with arithmetic coding. Because of the low interpixel dependencies, semiadaptive coding is more useful than adaptive coding, i.e. it is usually better to determine the accurate pixel frequencies beforehand, rather than during the actual coding.

The best overall compression was achieved by using *rla0b*. However, *rlh0* is faster and its compression factor is only slightly worse, while it is simpler to implement. The use of *rla0b* and *rlh0* improves compression generally at least 20 % when compared to *ewis*. By making some adjustments, the compression can be improved even further. For example, the repetition count n in run-length triplets (esc,n,ε) can be encoded more efficiently by using statistics gathered from previous occurrences of the triplets. Simply predicting the previous n and encoding the prediction error (using for example normal distribution) can improve the compression even dozens of bytes. Also note that since run-length triplets probably occur close to the position of a similar triplet in the previous row, the occurrence probability of *esc*-pixel in a certain row depends from the previous row(s). However, all these improvements also increase the execution time, as well as complicate the implementation.

In general, additional compression is possible for example through more efficient use of two-dimensional dependencies and adjustment of the coding probabilities. Pixel coding contexts (cloud area, borders, background) could be used more efficiently. However, *lossy compression* seems to be the only way to gain significant improvements. In lossy compression, information not needed is discarded so that the expanded file no longer exactly matches the original uncompressed file. It is probably better to use specific rather than general methods like (*lossy*) *jpeg* (*Wallace*, 1991), which can lose important information. This requires analysis of the observation data and its uses, as well as analysis of the noise introduced by the radar equipment, to find redundancies. For example, if we were not interested in small changes in the data, the pixels could be represented with fewer

bits (Table 6). Then the data would be more stabilized and more compressible. On the other hand, very specific algorithms may not be generally applicable anymore.

Table 6. Left column shows the amount of bits used to represent a pixel while the figures indicate the corresponding compression ratio of *rla0b*. Some redundancy is present since the coding algorithm wasn't modified to make use of the reduced pixel set. See Table 3 for eight bit results.

Bits	IHMXN	IPPIN	IC
7	9.83	10.85	28.82
6	13.74	12.30	33.00
5	16.61	14.33	39.68

Other factors than compression ratio should also be considered. Simple run-length algorithms are fast, and easy to implement. The compression factor is quite good. On the other hand, more advanced algorithms are usually slower, and tougher to implement, while the improvement in compression ratio is often marginal. Furthermore, general algorithms do not necessarily make use of the high amount of the background pixel present in weather radar data. Error detection and correction as well as compatible data format are also important in certain applications.

6. *References*

Abramson, N., 1963: Information Theory and Coding, *McGraw-Hill*, 77-81, USA.

Bell, T.C., Cleary, J.G. and Witten, I.H., 1990: Text Compression. *Prentice Hall Advanced Reference Series*, 47-48, USA.

Chang, C.Y. and Kwock, R., 1988: Spatial compression of Seasat SAR imagery. *IEEE Transactions on Geoscience and Remote Sensing* **26**, 5, 673-685.

Chen, T.M., Staelin, D.H. and Arps, R.B., 1987: Information content analysis of Landsat image data for compression. *IEEE Transactions on Geoscience and Remote Sensing* **25**, 4, 499-501.

Chevion, D., Karnin, E.D. and Walach, E., 1991: High efficiency, multiplication free approximation of arithmetic coding. Proceedings of Data Compression Conference, April 8-11, Snowbird, Utah, *IEEE Computer Society Press*, 43-52.

Cleary, J.G. and Witten, H.W., 1984: Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications* **32**, 4, 396-402.

Dietrich, E., Leonardi, R.M. and Sorani, R., 1990: BUFR code and data compression. Contribution to WG 1 - Telecommunications: Working document 73/wd/173, November.

Hadenfeldt, A.C. and Sayood, K., 1994: Compression of color-mapped images. *IEEE Transactions on Geoscience and Remote Sensing* **32**, 3, 534-541.

Howard, P.G. and Vitter, J.S., 1991: New methods for lossless image compression using arithmetic coding. Proceedings of Data Compression Conference, April 8-11, Snowbird, Utah, *IEEE Computer Society Press*, 257-266.

Howard, P.G. and Vitter, J.S., 1992: Error modelling for hierarchical lossless image compression. Proceedings of Data Compression Conference, March 24-27, Snowbird, Utah, *IEEE Computer Society Press*, 269-278.

Howard, P.G. and Vitter, J.S., 1992: Error modelling for hierarchical lossless image compression. Proceedings of Data Compression Conference, March 24-27, Snowbird, Utah, *IEEE Computer Society Press*, 269-278.

King, R.H., 1989: Operational experiences with the Finnish weather radar network. Weather Radar Networking, seminar on COST Project 73, September 5-8 Brussels, *Kluwer Academic Publishers*, 91-100.

King, R.H., 1993: Personal communications at FMI during 1993.

Lelewer, D.A. and Hirschberg, D.S., 1987: Data compression. *ACM Computing Surveys* **19**, 3, 261-296.

Leung, W-H.and Skiena, S.S., 1991: Inducing codes from examples (extended abstract). Proceedings of Data Compression Conference, April 8-11, Snowbird, Utah, *IEEE Computer Society Press*, 267-276.

Lu, G., 1993: Advances in digital image compression techniques. *Computer Communications* **16**, 4, 202-214.

Memon, N.D., Sayood, K. and Magliveras, S.S., 1994: Lossless compression of multispectral image data. *IEEE Transactions on Geoscience and Remote Sensing* **32**, 2, 282-289.

Nelson, M., 1991: The Data Compression Book. Prentice Hall, USA, 233-309.

Newsome, D.H. (ed.), 1992: Weather Radar Networking. COST Project 73, Final Report, *Kluwer Academic Publishers*, 95-99.

Näppi, J., 1994: Säätutkahavaintojen tiivistys. Master's Thesis, University of Helsinki, Department of Computer Science.

Puhakka, T.M. and Sarvi, A.A., 1979: On the compression of digital radar data. *Geophysica* **16**, 1, 81-96.

Reghbati, H.K., 1981: An overview of data compression techniques. *Computer* **14**, 4, 71-75.

Rissanen, J., Langdon Jr., G.G., 1979: Arithmetic coding. *IBM Journal of Research and Development* **23**, 2, 149-162.

Rissanen, J., Mohiuddin, K.M., 1989: A multiplication-free multialphabet arithmetic code. *IEEE Transactions on Communications* **37**, 2, 93-98.

Samet, H., 1984: The quadtree and related hierarchical data structures. *Computing Surveys* **16**, 2, 187-260.

Sandstroem, M., 1986: Ericsson Radio Systems, EWIS document HL/Sir: *Compression format.* Sheets 2-3.

Shannon, C.E., 1948: A mathematical theory of communication. *Bell System Technical Journal* **28**, 379-423.

Wallace, G.K., 1991: The JPEG still picture compression standard. *Communications of the ACM* **34**, 4, 31-44.

Witten, I.H., Radford, M.N. and Cleary, J.G., 1987: Arithmetic coding for data compression. *Communications of the ACM* **30**, 6, 520-540.

Ziv, J. and Lempel, A., 1977: A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* IT-23, 3, 337-343.

Ziv, J. and Lempel, A., 1978: Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory* IT-24, 5, 530-536.